

H. Blocs imbriqués & portée d'une variable

Définition : Bloc imbriqué

Un bloc PLSQL imbriqué est un bloc déclaré à l'intérieur du bloc **BEGIN** d'un autre bloc PLSQL.

Syntaxe :

DECLARE

.....

BEGIN

/* instructions */

.....

DECLARE

.....

BEGIN

/* instructions */

.....

END ;

END ;

Bloc imbriqué

Exemple

DECLARE

/* définition du type RECORD nommé : "product" */

TYPE product **IS** **RECORD** (

id "_produit"."id"%TYPE **NOT** **NULL** := 1,

designation "_produit"."designation"%TYPE,

qteStock "_produit"."qteStock"%TYPE,

prixUnit "_produit"."prixUnit"%TYPE

);

/* déclaration d'un product nommé produit_1 */

produit_1 product;

BEGIN

BEGIN

/* initialisation des attributs du product produit_1 */

produit_1.designation := 'PC Portable';

produit_1.qteStock := 5;

produit_1.prixUnit := 7000;

END;

dbms_output.put_line('ID produit: ' || produit_1.id);

dbms_output.put_line('Désignation: ' || produit_1.designation);

dbms_output.put_line('Qté stock: ' || produit_1.qteStock);

dbms_output.put_line('Prix unitaire: ' || produit_1.prixUnit);

END;

Définition : Portée d'une variable

La portée d'une variable est définie par les limites du bloc PLSQL dans lequel elle a été définie, et les sous-blocs de ce bloc. En dehors de ces limites, la variable est **inutilisable**. On parle alors de **variable locale**.

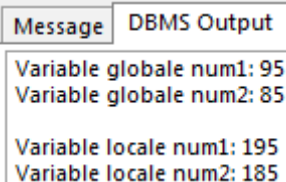
Si une variable est définie dans le bloc parent, elle est utilisable dans tous les blocs imbriqués. On dit alors que c'est une **variable globale**.

Dans un bloc imbriqué, si une variable locale a le même nom qu'une variable globale,

alors cette dernière est éclipsee jusqu'à ce que le bloc imbriqué soit terminé.

Exemple 1

```
DECLARE
  -- Variables globales
  num1 number := 95;
  num2 number := 85;
BEGIN
  dbms_output.put_line('Variable globale num1: ' || num1);
  dbms_output.put_line('Variable globale num2: ' || num2);
  DECLARE
    -- Variables locales
    num1 number := 195;
    num2 number := 185;
  BEGIN
    dbms_output.new_line();
    dbms_output.put_line('Variable locale num1: ' || num1);
    dbms_output.put_line('Variable locale num2: ' || num2);
  END;
END;
```



Message DBMS Output

Variable globale num1: 95
Variable globale num2: 85

Variable locale num1: 195
Variable locale num2: 185

Exemple 2

```
DECLARE
  -- Variables globales
  num1 number := 95;
  num2 number := 85;
BEGIN
  dbms_output.put_line('Variable globale num1: ' || num1);
  dbms_output.put_line('Variable globale num2: ' || num2);
  DECLARE
    -- Variables locales
    a number := 195;
    b number := 185;
  BEGIN
    dbms_output.new_line();
    dbms_output.put_line('Variable globale num1: ' || num1);
    dbms_output.put_line('Variable globale num2: ' || num2);
  END;
  dbms_output.put_line('Variable locale: ' || a);
  dbms_output.put_line('Variable locale: ' || b);
END;
```

Erreur. Variable utilisée en dehors de sa portée

II. Les structures de contrôle

A. Expressions booléennes et opérateurs

Définition : Expression booléenne													
Une expression est dite booléenne quand sa valeur globale est soit TRUE soit FALSE. Une expression booléenne peut contenir des opérateurs relationnels, logiques, ou de comparaison.													
Définition : Opérateurs relationnels													
=	Egal												
!= ou bien <> ou bien ~=	Différent de												
>	Supérieur												
<	Inférieur												
>=	Supérieur ou égal												
<=	Inférieur ou égal												
Définition : Opérateurs de comparaison													
LIKE	<p>Comparaison entre chaînes de caractères. Peut être utilisé avec les opérateurs <code>_</code> et <code>%</code></p> <table> <tr> <td>'bonjour' LIKE 'on'</td><td>false</td></tr> <tr> <td>'bonjour' LIKE '%on%'</td><td>true</td></tr> <tr> <td>'bonjour' LIKE '_on%'</td><td>true</td></tr> <tr> <td>'bonjour' LIKE '_on_'</td><td>true</td></tr> <tr> <td>'bonjour' LIKE 'bonj_ur'</td><td>true</td></tr> <tr> <td>'bonjour' LIKE 'bonjou_'</td><td>false</td></tr> </table> <p>4 caractères <code>_</code> pour remplacer la séquence 'jour'</p>	'bonjour' LIKE 'on'	false	'bonjour' LIKE '%on%'	true	'bonjour' LIKE '_on%'	true	'bonjour' LIKE '_on_'	true	'bonjour' LIKE 'bonj_ur'	true	'bonjour' LIKE 'bonjou_'	false
'bonjour' LIKE 'on'	false												
'bonjour' LIKE '%on%'	true												
'bonjour' LIKE '_on%'	true												
'bonjour' LIKE '_on_'	true												
'bonjour' LIKE 'bonj_ur'	true												
'bonjour' LIKE 'bonjou_'	false												
BETWEEN ... AND ...	<p>Comparaison entre chiffres.</p> <pre>X := 15; X BETWEEN 11 AND 16</pre> <p>true</p>												
IN	<p>Teste si une valeur (chiffres ou caractères) figure parmi un ensemble de valeurs.</p> <pre>'x' IN ('a', 'b', 'c')</pre> <p>false</p> <pre>15 IN (0, 25, 15, 33)</pre> <p>true</p>												
IS NULL	Retourne true si la variable contient la valeur NULL .												
Définition : Opérateurs logiques													
AND	Le ET logique												
OR	Le OU logique												
NOT	Le NON logique												

B. Les structures alternatives

1. IF - THEN

Syntaxe

```
IF condition_booléenne THEN
    /* instructions */
END IF;
```

Exemple

```
DECLARE
    ch VARCHAR2(50);
BEGIN
    --test 1
    ch := 'bonjour vinci';
    IF ch LIKE '%jour v%' THEN
        dbms_output.put_line('1ier test vrai');
    END IF;
    --test 2
    IF ch LIKE '_jour v%' THEN
        dbms_output.put_line('2ieme test vrai');
    END IF;
    --test 3
    IF ch LIKE '%jour v____' THEN
        dbms_output.put_line('3ieme test vrai');
    END IF;
    IF ch LIKE '%jo% vin_i' THEN
        dbms_output.put_line('4ieme test vrai');
    END IF;
END;
```

Message DBMS Output

1ier test vrai
3ieme test vrai
4ieme test vrai

2. IF – THEN – ELSE

Syntaxe

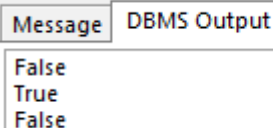
```
IF condition_booléenne THEN
    /* instructions */
ELSE
    /* autres instructions */
END IF;
```

Exemple 1

```
DECLARE
    c CONSTANT CHAR := 'm';
BEGIN
    IF (c IN ('a', 'b', 'c')) THEN
        dbms_output.put_line('True');
    ELSE
        dbms_output.put_line('False');
    END IF;

    IF (c IN ('m', 'n', 'o')) THEN
        dbms_output.put_line('True');
    ELSE
        dbms_output.put_line('False');
    END IF;

    IF (c is null) THEN
        dbms_output.put_line('True');
    ELSE
        dbms_output.put_line('False');
    END IF;
END;
```



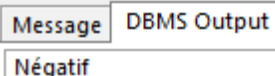
Message DBMS Output

False
True
False

Exemple 2

Déclarer et initialiser 2 chiffres, puis afficher le signe (positif / négatif) de leur produit, sans calculer le produit. (ignorer le cas de 0)

```
DECLARE
    x CONSTANT NUMBER := -2;
    y CONSTANT NUMBER := 2;
BEGIN
    IF (x > 0 AND y > 0) OR (x < 0 AND y < 0) THEN
        dbms_output.put_line('Positif');
    ELSE
        dbms_output.put_line('Négatif');
    END IF;
END;
```



Message DBMS Output

Négatif

3. IF imbriqué

Définition

On dit qu'un bloc **IF** est imbriqué, s'il est lui-même défini à l'intérieur d'un autre bloc **IF**, ou **ELSE**.

Syntaxe :

```
IF condition_booléenne THEN
    /* instructions */
    IF condition_booléenne THEN
        /* instructions */
    END IF;
ELSE
    /* autres instructions */
END IF;
```

Ou :

```
IF condition_booléenne THEN
    /* instructions */
ELSE
    /* autres instructions */
    IF condition_booléenne THEN
        /* instructions */
    END IF;
END IF;
```

Exemple

Refaire l'exemple précédent en utilisant 3 chiffres. (ignorer le cas de 0)

```
DECLARE
    x CONSTANT NUMBER := 2;
    y CONSTANT NUMBER := -1;
    z CONSTANT NUMBER := 5;
BEGIN
    IF (x > 0 AND y > 0) OR (x < 0 AND y < 0) THEN
        IF z > 0 THEN
            dbms_output.put_line('Positif');
        ELSE
            dbms_output.put_line('Négatif');
        END IF;
    ELSE
        IF z > 0 THEN
            dbms_output.put_line('Négatif');
        ELSE
            dbms_output.put_line('Positif');
        END IF;
    END IF;
END;
```

Message DBMS Output
Négatif

Définition : ELSIF

Le bloc **ELSIF** a le même effet qu'un bloc **IF** imbriqué dans un bloc **ELSE**. Mais il surtout est plus pratique quand on veut faire des tests sur des valeurs multiples.

Syntaxe :

```
IF(booléen_1) THEN
    instruction_1;
ELSIF(booléen_2) THEN
    instruction_2;
ELSIF(booléen_3) THEN
    instruction_3;
ELSE
    /*
        sera exécutée si booléen_1, booléen_2 et booléen_3 ne sont pas TRUE
    */
    instruction_4;
END IF;
```

Exemple

```
DECLARE
    x CONSTANT VARCHAR2(1) := 'B';
BEGIN
    IF x = 'A' THEN
        dbms_output.put_line('Excellent !');
    ELSIF x = 'B' THEN
        dbms_output.put_line('Très bien');
    ELSIF x = 'C' THEN
        dbms_output.put_line('Bien');
    ELSIF x = 'D' THEN
        dbms_output.put_line('Moyen');
    ELSIF x = 'E' THEN
        dbms_output.put_line('Passable');
    ELSE
        dbms_output.put_line('Il vaut mieux redoubler !!');
    END IF;
END;
```

Message DBMS Output

Très bien

4. CASE

Définition

L'opérateur **CASE** sert à faire un choix entre plusieurs valeurs possibles d'une variable (comme l'exemple précédent).

Même si on peut faire la même chose avec l'opérateur **IF**, le **CASE** est une manière plus facile et plus élégante.

1^{ère} Syntaxe :

```
CASE nom_variable
  WHEN 'valeur1' THEN instruction_1;
  WHEN 'valeur2' THEN instruction_2;
  WHEN 'valeur3' THEN instruction_3;
  ...
  ELSE instruction_n;
END CASE;
```

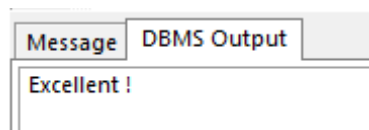
2^{ème} Syntaxe :

```
CASE
  WHEN nom_variable = 'valeur1' THEN instruction_1;
  WHEN nom_variable = 'valeur2' THEN instruction_2;
  WHEN nom_variable = 'valeur3' THEN instruction_3;
  ...
  ELSE instruction_n;
END CASE;
```

Exemple 1

Refaire l'exemple précédent en utilisant l'opérateur **CASE** 1^{ère} Syntaxe

```
DECLARE
  x CONSTANT VARCHAR2(1) := 'A';
BEGIN
  CASE x
    WHEN 'A' THEN
      dbms_output.put_line('Excellent !');
    WHEN 'B' THEN
      dbms_output.put_line('Très bien');
    WHEN 'C' THEN
      dbms_output.put_line('Bien');
    WHEN 'D' THEN
      dbms_output.put_line('Moyen');
    WHEN 'E' THEN
      dbms_output.put_line('Passable');
    ELSE
      dbms_output.put_line('Il vaut mieux redoubler !!');
  END CASE;
END;
```



Exemple 2

Refaire l'exemple précédent en utilisant l'opérateur **CASE** 2^{ème} Syntaxe

```
DECLARE
  x CONSTANT VARCHAR2(1) := 'A';
```



```

BEGIN
  CASE
    WHEN x = 'A' THEN
      dbms_output.put_line('Excellent !');
    WHEN x = 'B' THEN
      dbms_output.put_line('Très bien');
    WHEN x = 'C' THEN
      dbms_output.put_line('Bien');
    WHEN x = 'D' THEN
      dbms_output.put_line('Moyen');
    WHEN x = 'E' THEN
      dbms_output.put_line('Passable');
    ELSE
      dbms_output.put_line('Il vaut mieux redoubler !!');
  END CASE;
END;

```



IMPORTANT

Un bloc appartenant à un opérateur **IF ELSE ELIF** ou **CASE** peut contenir plusieurs instructions, y compris des blocs **IF ELSE ...**

Exemple

```

DECLARE
  x CONSTANT VARCHAR2(1) := 'A';
  note CONSTANT NUMBER(3,1) := 19.5;
BEGIN
  CASE
    WHEN x = 'A' THEN
      IF note >= 19.5 THEN
        dbms_output.put_line('Excellent !');
        dbms_output.put_line('Excellent !');
        dbms_output.put_line('Excellent !');
      ELSE
        dbms_output.put_line('Excellent !');
      END IF;
    WHEN x = 'B' THEN
      dbms_output.put_line('Très bien');
    WHEN x = 'C' THEN
      dbms_output.put_line('Bien');
    WHEN x = 'D' THEN
      dbms_output.put_line('Moyen');
    WHEN x = 'E' THEN
      dbms_output.put_line('Passable');
    ELSE
      dbms_output.put_line('Il vaut mieux redoubler !!');
  END CASE;
END;

```

Message DBMS Output

Excellent !
Excellent !
Excellent !